

No-Code AI Avatar that Remembers Your Notes and Answers Questions with its Own Voice using GPT-3, Siri and ElevenLabs



Avatar

Both ChatGPT and GPT-3 are great tools if used wisely. By writing good prompts, you can unlock their true potential. But there is something more, which is an access to the Internet and knowledge about YOU! And the fun part is, that you can achieve this with no-code tools like [Airtable](#), [Make.com](#) and [Pinecone](#) (vector database).

In a previous post, [I shared my vision of an AI Assistant](#). In my other post, you can find a detailed implementation in [Node.js](#). This time, I will show you how to do the same with a single line of code (rly!).

Required Apps & Services

Our AI Avatar needs to have a memory. In our case, we will use [Airtable](#); its API and UI make it simple and easy to use. In a database we need two tables: Memory (for storing long-term data) and Conversation (for storing a current conversation).

	record_id	content	tags	source	sync	Forget
1	recHfAEuUJK...					Forget
2	recaY9JruASit...					Forget
3	rec3Wf1zdJ6Q...					Forget
4	recME12eP4rF...					Forget

Memory

	record_id	question	answer	minutes	summary	created_at
+						

Conversation

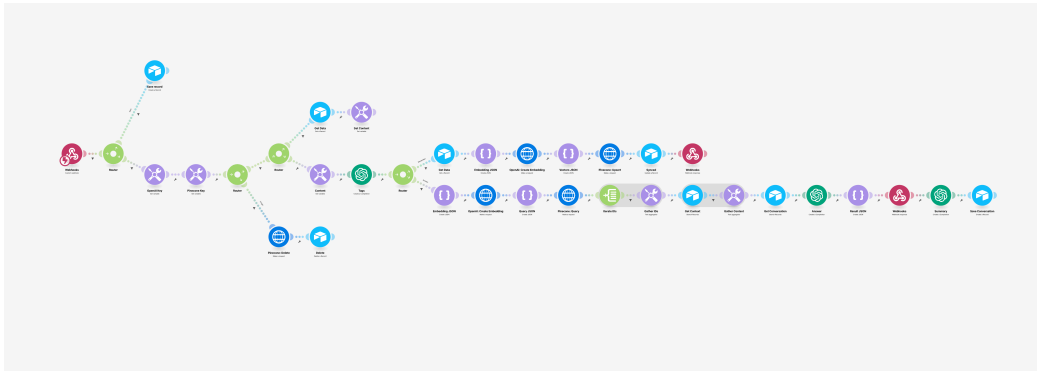
Note: At some point, there will be a need to use the premium plan of Airtable.

While storing information is crucial, there is something even more important: searching. In theory, we could use a search engine such as Algolia or ElasticSearch, but configuration would be much more difficult and would require some coding. To make it really easy and effective, we'll gonna use a Vector Database called Pinecone. It will perform something called a "similarity search". So, every time we ask Avatar a question, Pinecone will search for meaningful information, based on which we will get a response.

Do not worry! For now, there is like nothing more you need to know about this. We just need a three, easy API calls to get through this. I'll show you that in a minute.

Note: Pinecone has a good free plan for learning & small projects. At some point there may be a need to use its paid version.

A third app is Make.com. If you don't yet know about it, #withMake you can easily connect to hundreds of services and even any custom API. You need a minimum Core or Pro plan of Make to get started. Avatar's main logic will be built in a scenario such as this:



Mind

I assume that you have at least basic experience with make, but I'll try to stay as detailed as it possible.

And the last one is GPT-3 API, which is a paid service! You should **definitely** set a hard limit and you're using those tools at your own risk. Regardless of whether you use Node.js or Make.com, there is a chance of making too many API calls for which you will have to pay real money. Keep that in mind, because a hard limit is our only hope (ha. ha.).

To put all of this together, overall flow looks like this:

1. make.com is logic
2. Pinecone is smartness
3. Airtable is memory
4. GPT-3 is mouth
5. (bonus) ElevenLabs is voice

When we speak with Avatar it will:

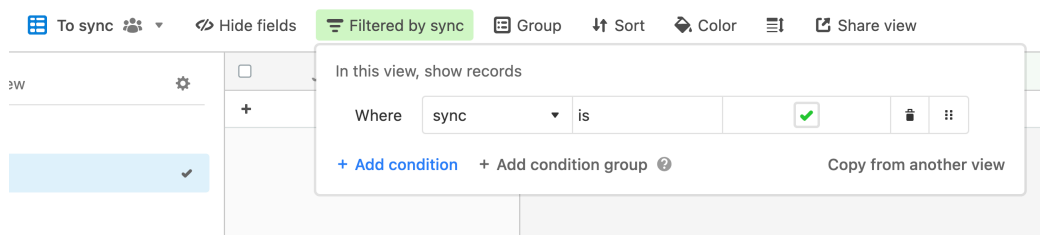
1. Connect with a make.com scenario
2. Fetch relevant information with Pinecone
3. Get actual data from Airtable
4. Generate a response with GPT-3
5. And return it with make (optionally, it may be sent to EL to convert text to speak)

Long term & Short term memory

Let's deep dive into Airtable database. As you can see on a screenshots above, we have a couple of columns and views.

Memory table

In the memory table, we have two views: "All" and "To Sync". The first one includes all of the records we have, and the second filters those in which the "Sync" column is checked:



Memory table includes columns:

- record_id (type formula with a value: RECORD_ID()),
- content (Long text),
- source (Single line),
- Sync (Checkbox),
- Forget (Button that will redirect us to the webhook. We'll back to this later)

In this table, we'll store all of the information that our Avatar will have access to.

Conversation table

Second table is really simple:

- record_id (type formula with a value: RECORD_ID()),
- question (Long text),
- answer (Long text),
- summary (Long text),
- minutes (Formula: DATETIME_DIFF(CREATED_TIME(), NOW(), "minutes"))
- created_at (Created Time)

In those columns, we'll store a current conversation and summaries.

Smart Search (Similarity Search)

The next step is to setup a database in Pinecone and this is really simple. Create a free account and use settings like this:

Create Index

NEW

FROM COLLECTION

Index Name*

avatar

Dimensions*

1536

Metric*

cosine

Pod Type

S1

Best storage capacity

P1

Faster queries

P2

Lowest latency and highest throughput

ADVANCED CONFIGURATION

Starter Plan Index

UPGRADE

Indexes in the Starter Plan will be terminated after 14 days of inactivity

CREATE INDEX

CANCEL

Hit "Create Index" and then copy two values: the index URL and your API key. Save them to your notes, as we'll need them later.

Index Name

monday

monday- [progress bar] [progress bar] [progress bar] [progress bar] cone.io

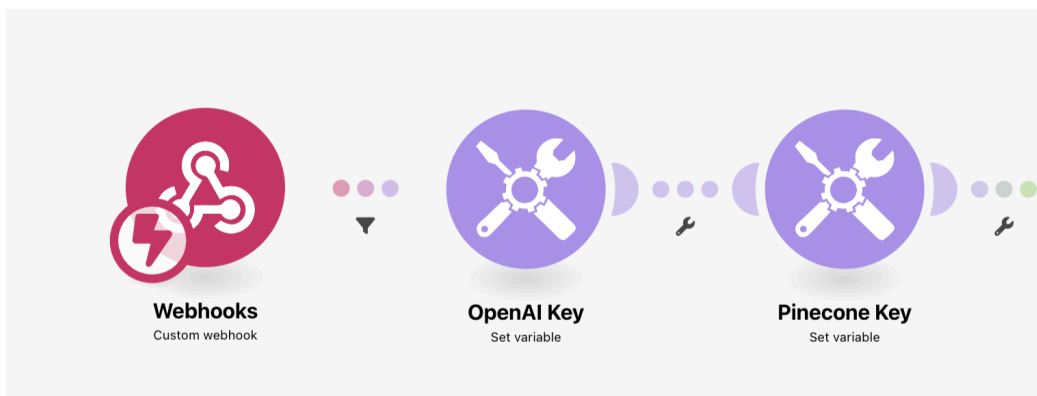
● Ready

Believe this or not, but now you can close Pinecone, because its API is all we need.

Initialize Logic

It's time to create a new scenario in [make.com](#). It needs to be triggered with a Webhook, which will be listening for HTTP POST requests with a JSON body.

First three modules like this:



Webhook will accept POST HTTP requests with a JSON body like this:

```
{
  "content": "[query]",
  "record_id": "recXXXXXXX"
  "type": "ask" // forget / save / remember
}
```

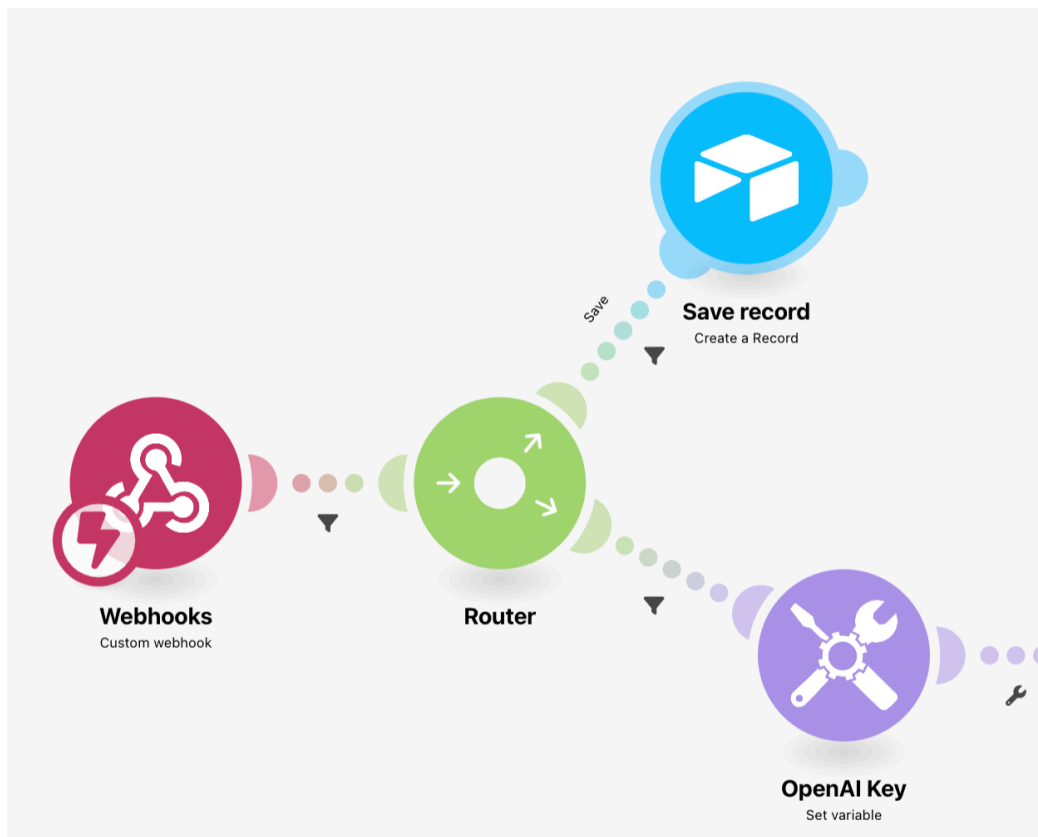
Content is a query from us; it may be information to be saved or answered to, depending on the `type` property. The `record_id` will be used only for Airtable interactions, and the `type` property describes which action a scenario needs to take. Everything will be clear in a moment.

Two additional modules are actions "Set Variable" from "Tools" module (it's native make's module). As values of those variable, provide both names like `openai_key` and `pinecone_key` and then paste your API keys from OpenAI and Pinecone.

Storing information in Airtable

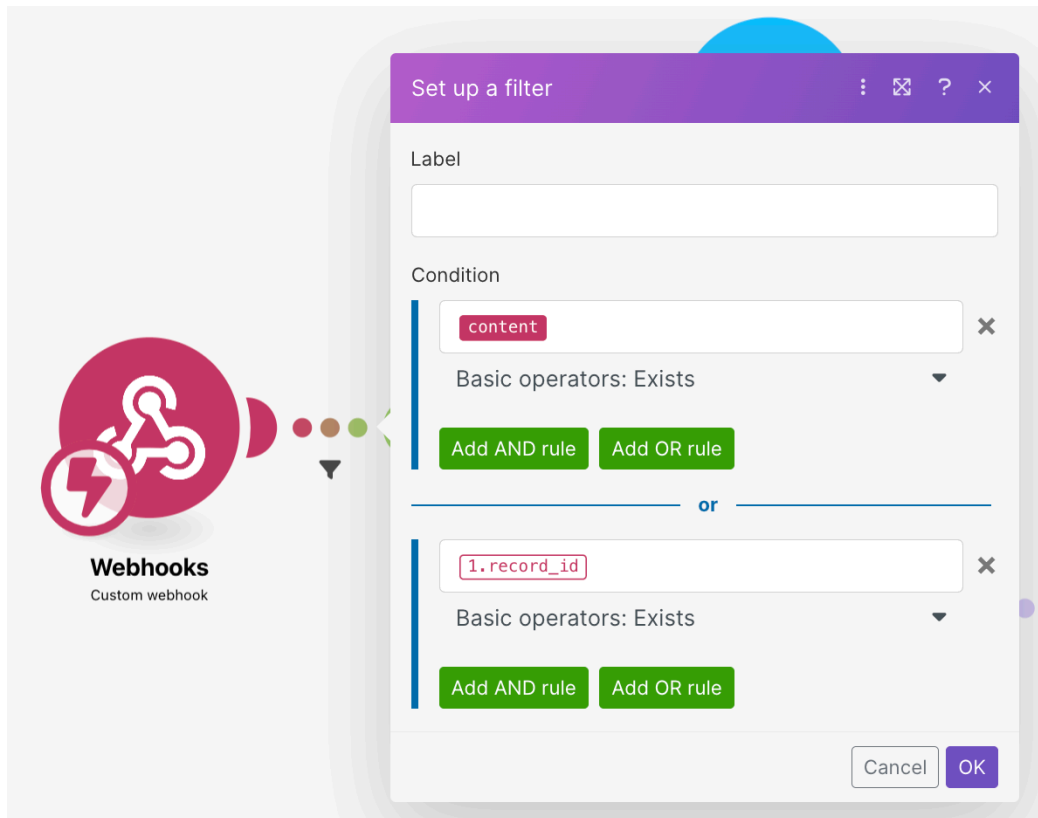
There is a difference between `storing` (in Airtable) and `memorizing` (in Pinecone). Because of that, we will split this into two steps.

Storing information will require us to modify our scenario by adding a "route" like so:



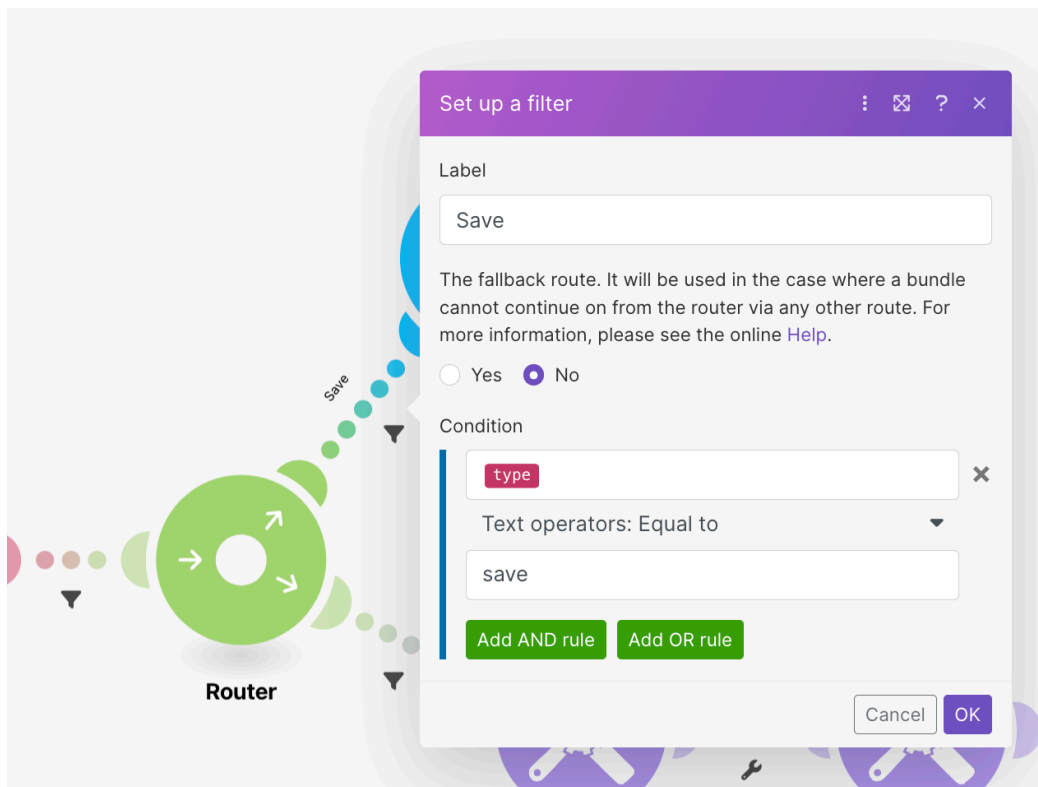
As you can see, between Webhooks and OpenAI Key, I have added a Router module and connected it to both the Create a Record action from Airtable and the rest of the scenario.

There are some important details here: filters. If you click on chains between modules, you'll see a "Filter" option. First one is this between Webhook and Router and looks like this:

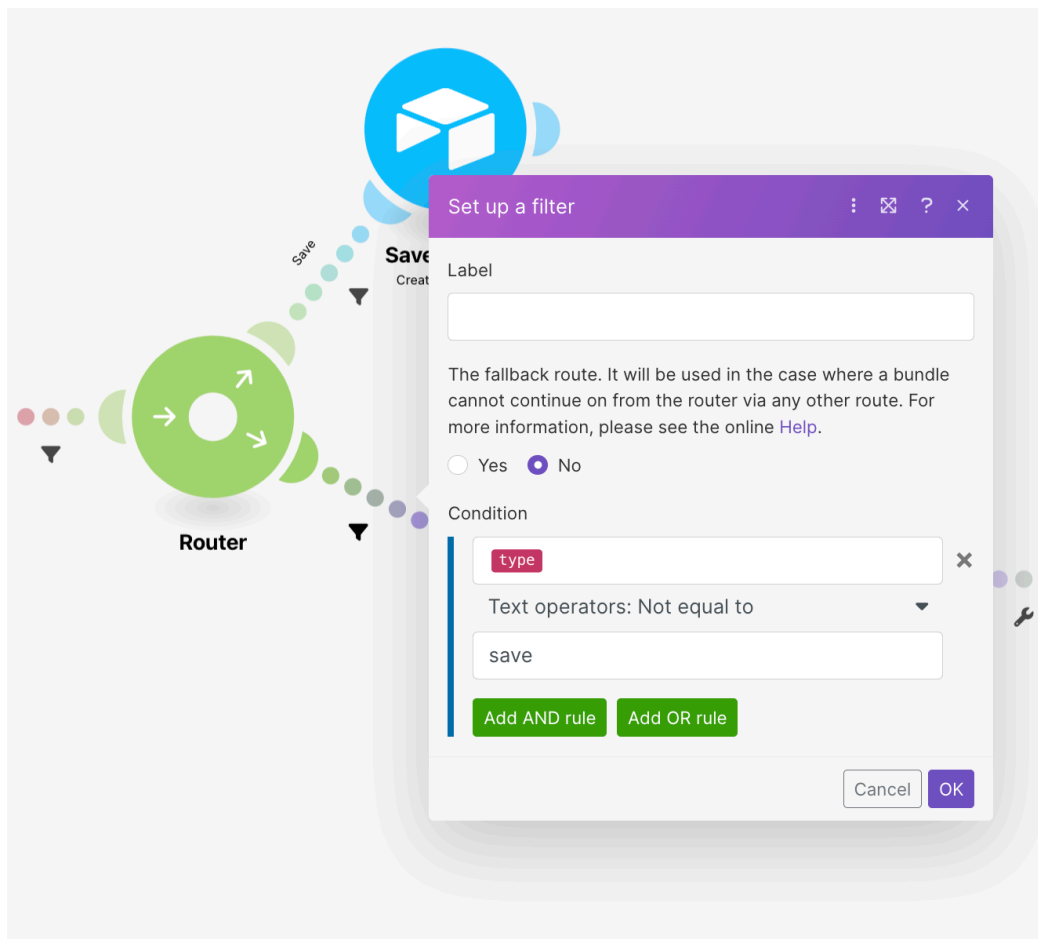


As you can see I'm checking here if there is a `content` or `record_id` variable available in a payload.

A second and third filters checks if `type` is set to "save" **or to some other value**:



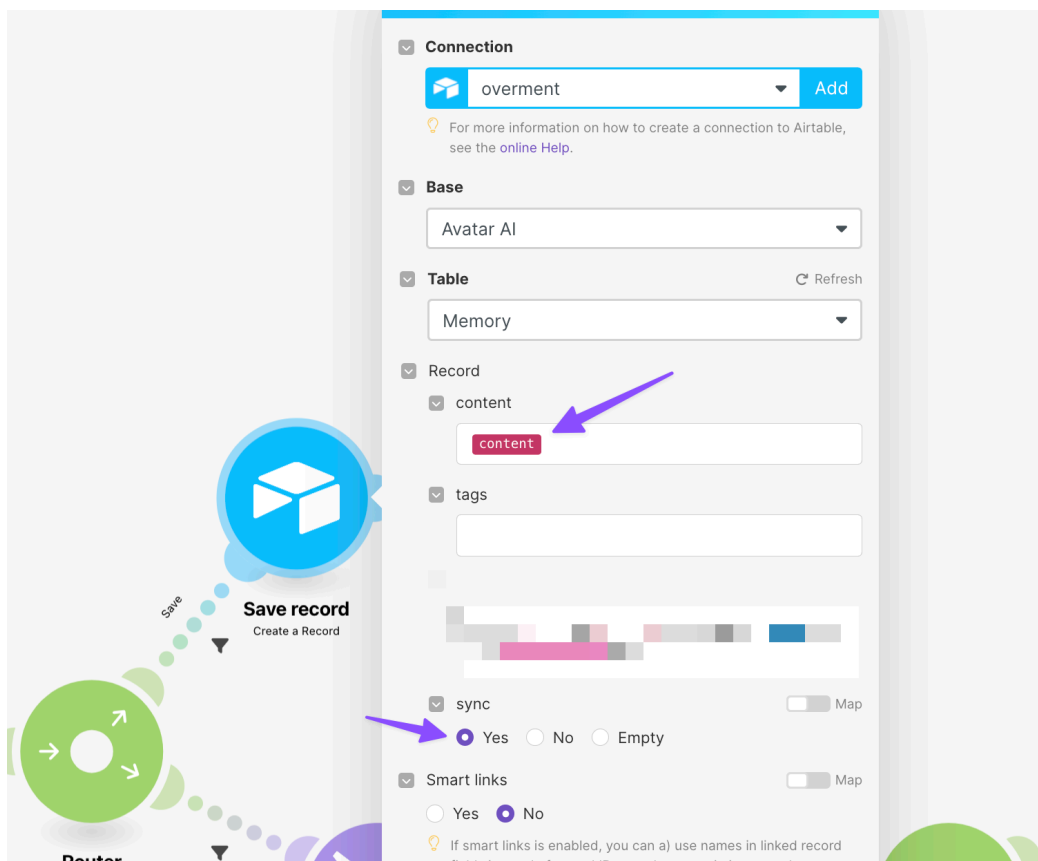
First route: Type = save



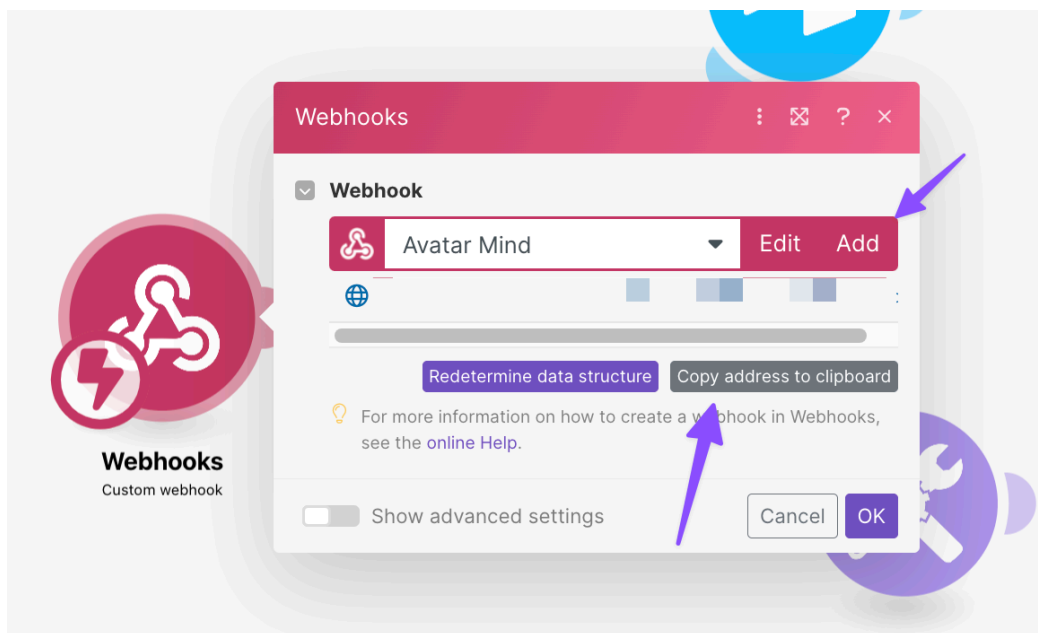
Second route: Type != save

It's really important to make sure that those filters are set properly.

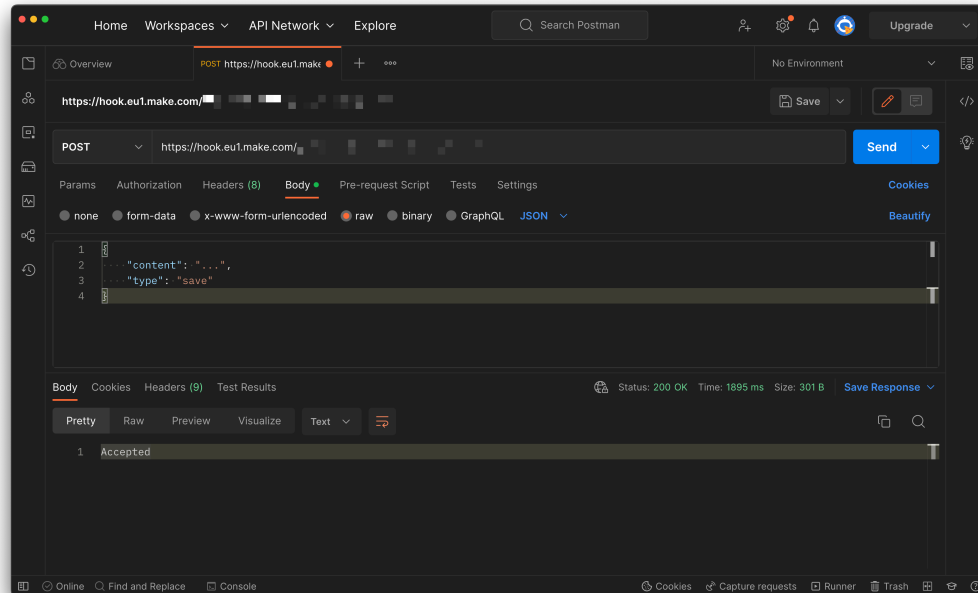
If a type is set to "save", we need to store informations in Airtable like below. Just double check if `sync` column is set to "Yes" (you can ignore blurred part from a screenshot below).



Now you can use any HTTP client to make a POST request to your Webhook. Just copy its address. Then save a whole [make.com](https://www.make.com) scenario and activate it.



If you send a JSON with properties `content` and `type` (save), an information should appear in Airtable.



Postman HTTP Request

Airtable Automation

Any information that our Avatar needs to remember, will be first saved in Airtable with "sync" checkbox selected. Because of this, this record will go into "To Sync" view, we have created. Such event may be a trigger for Airtable's built-in automations.

I have created an automation which triggers a JavaScript snippet (as mentioned, it's just a single line of code!):

```
// Replace WEBHOOK_ID with your own
fetch(`https://hook.eu1.make.com/WEBHOOK_ID?
type=remember&record_id=${input.config().id}`)
```




All you need to do is to make sure that automation is connected with "To Sync" view, "id" variable is connected with "record_id" field and it is active.

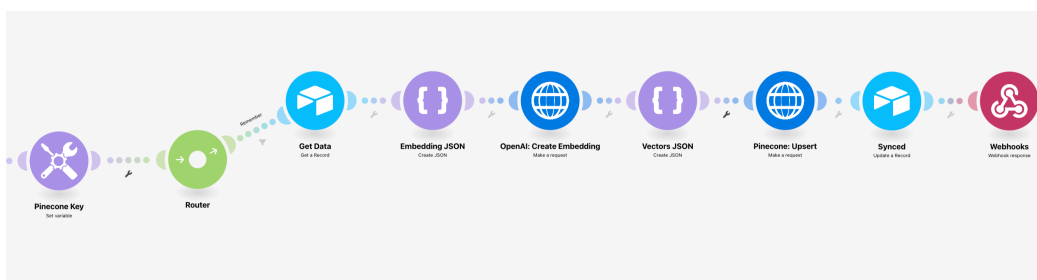
An idea here is a very simple one. Every time a record will appear in **To Sync** view, it will trigger a JavaScript snippet, which will send an identifier of a record to the scenario in make.com.

If you trigger HTTP request in Postman once again, you'll see that your scenario in make.com will be triggered twice: first because of a request you've just sent and a second one because of Airtable automation.

What's more, if you need to update information already saved in Airtable, you can just select "sync" checkbox and wait a couple of seconds. At this point nothing else will happen, because we don't have a proper logic yet.

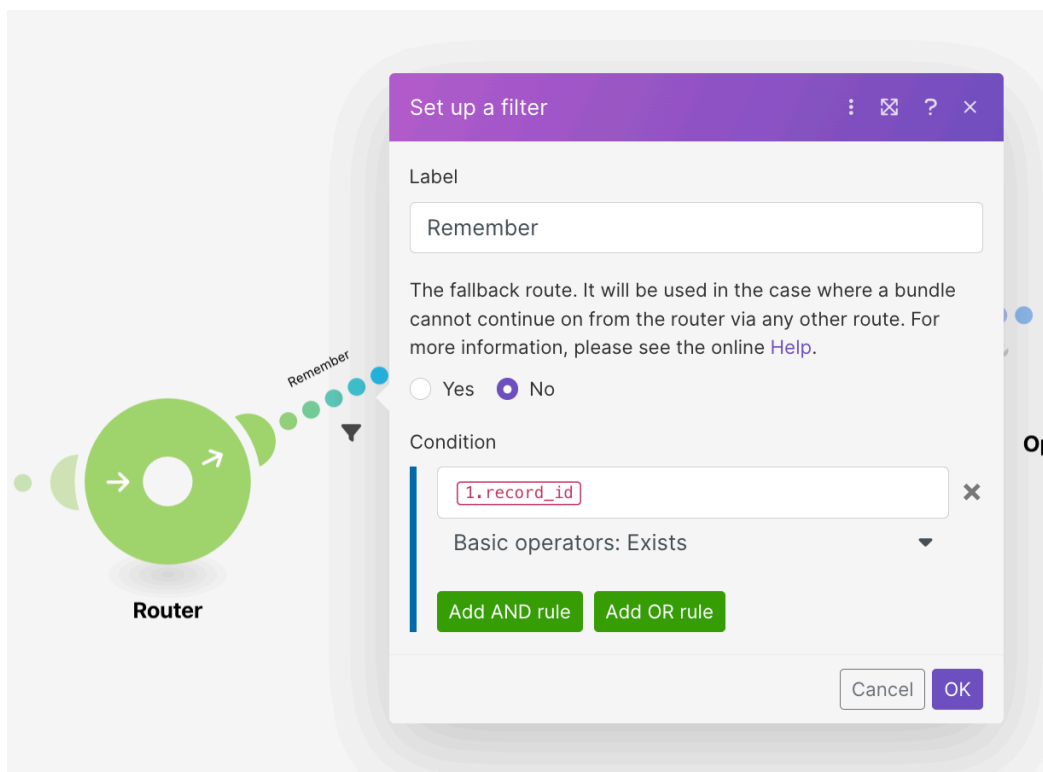
Airtable & Pinecone Sync

We're ready to sync records with Pinecone. Right after the last "Set Variable" module, we'll need additional modules.

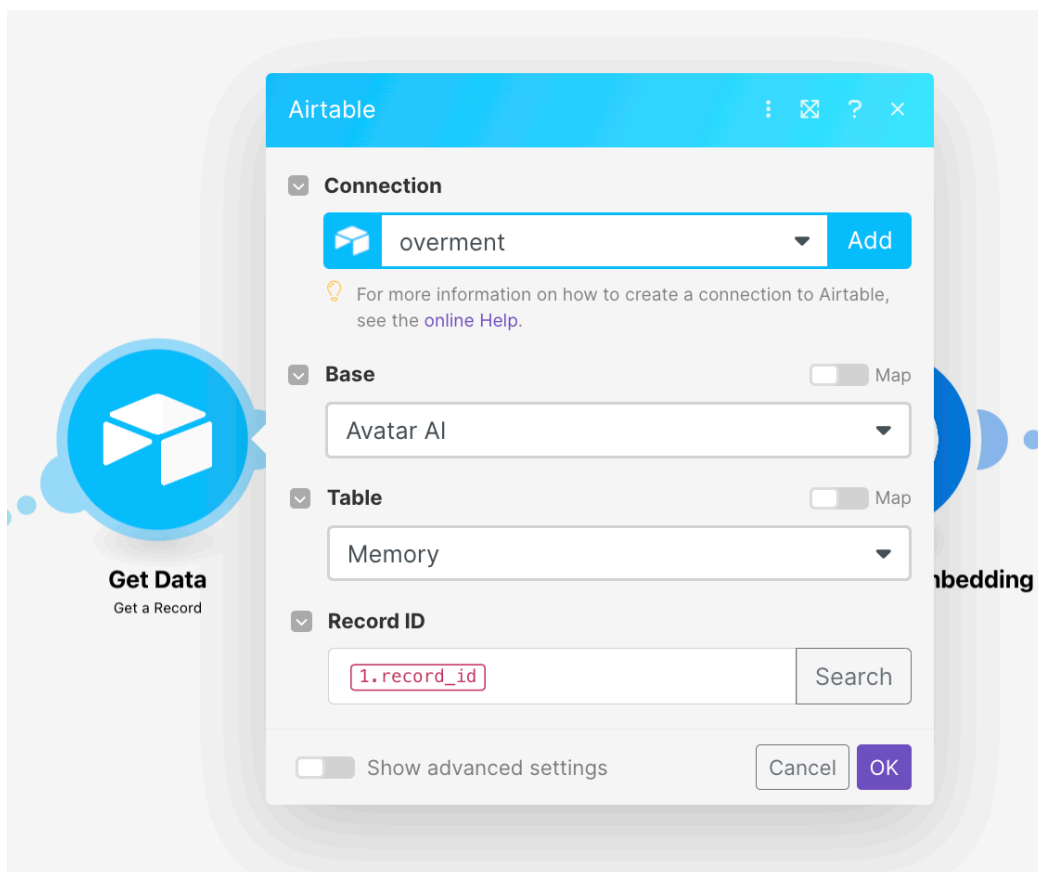


Let's explain each of them.

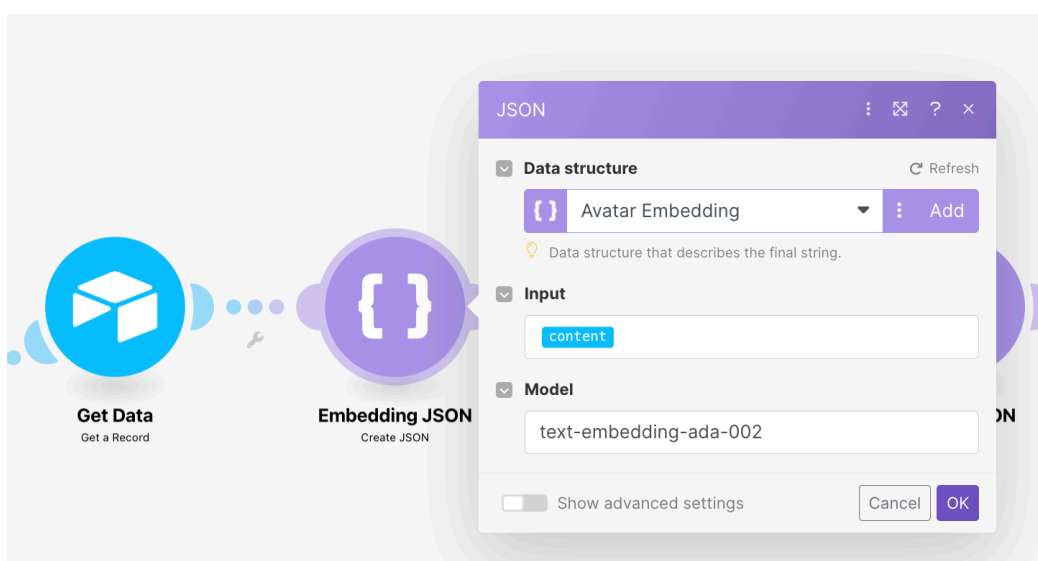
1. Router for splitting the path and filter to verify if `record_id` variable is present in JSON from Webhook.



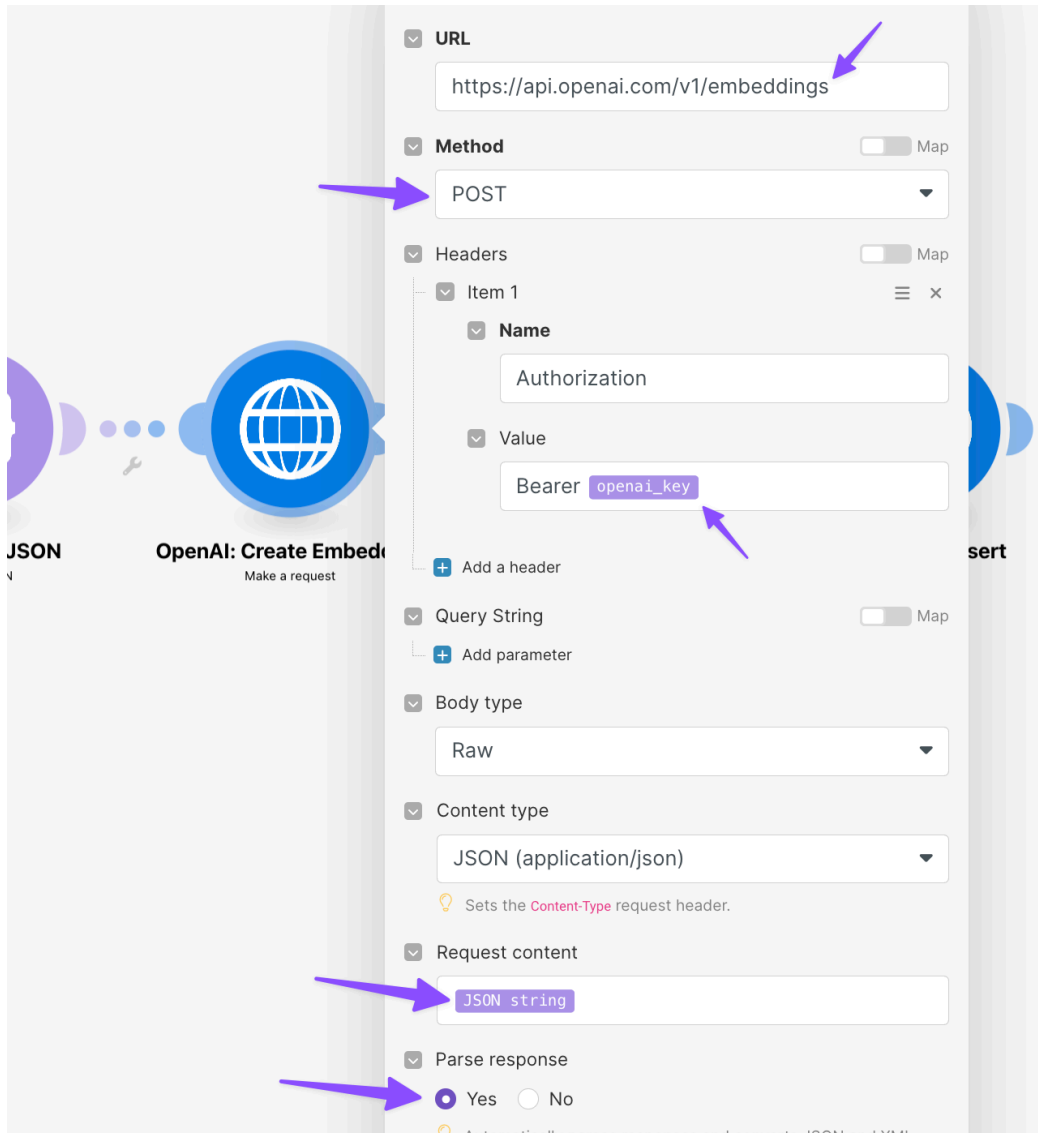
1. We need fetch additional details of this record from Airtable. We can do this with action `Get a Record` from Airtable module.



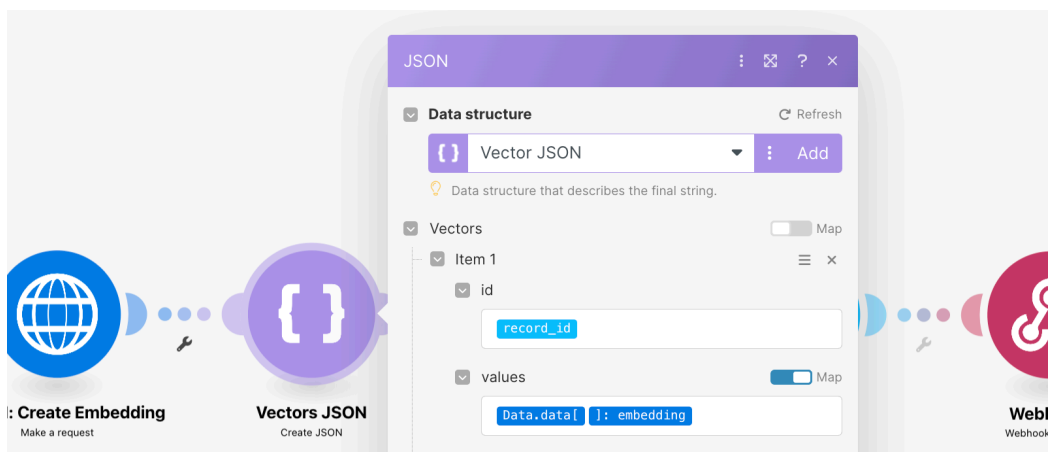
1. In the next step, we need to create a JSON in the JSON module. To do this, first you have to "Add" a Data Structure, which you can generate by pasting a JSON object like this: `{"input": "", "model": ""}`. Then, add the content field from Airtable and set the Model to `text-embedding-ada-002`.



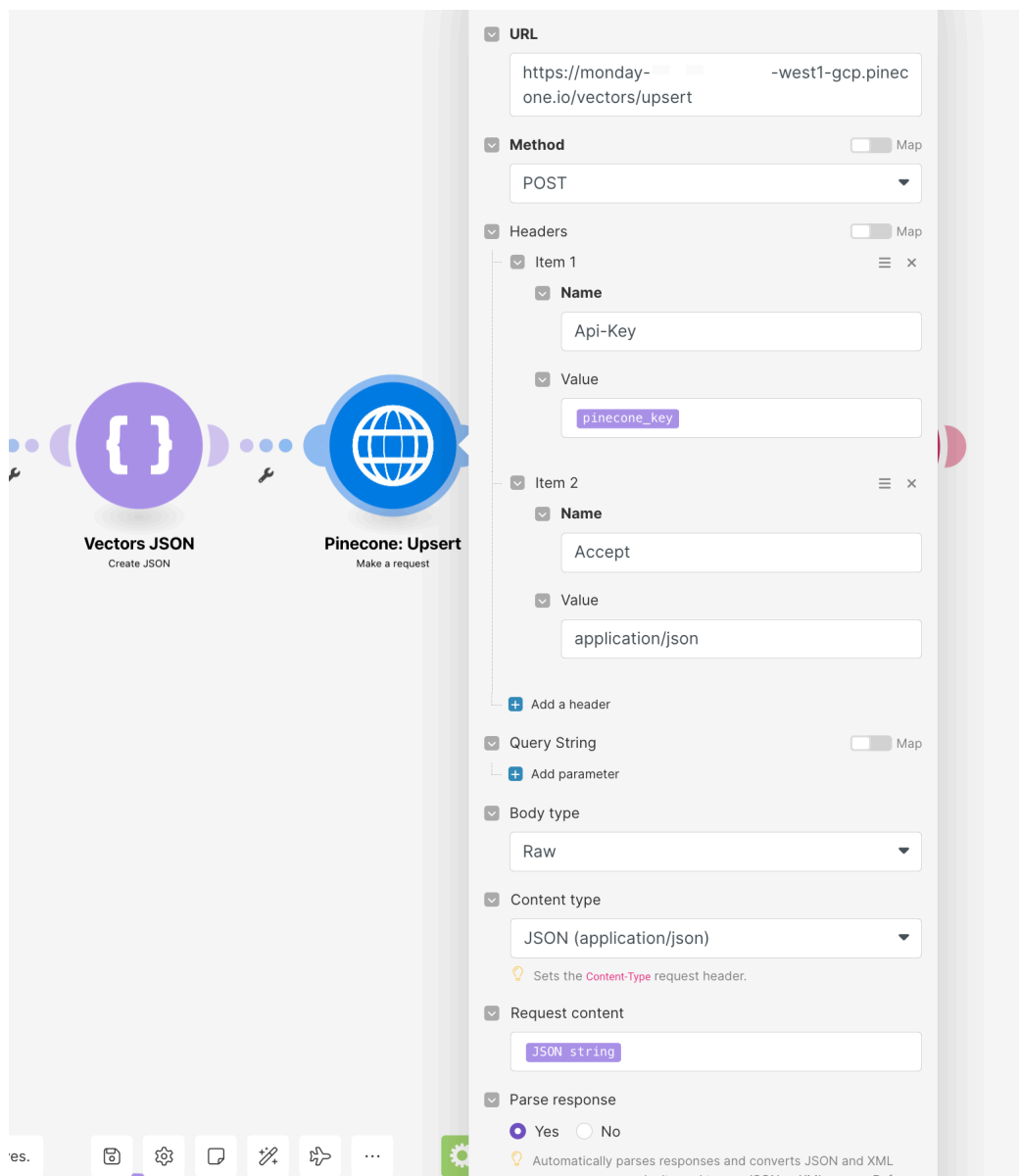
1. Created JSON needs to be passed with an HTTP request to OpenAI, because we need to convert the information from Airtable to something called "embeddings".



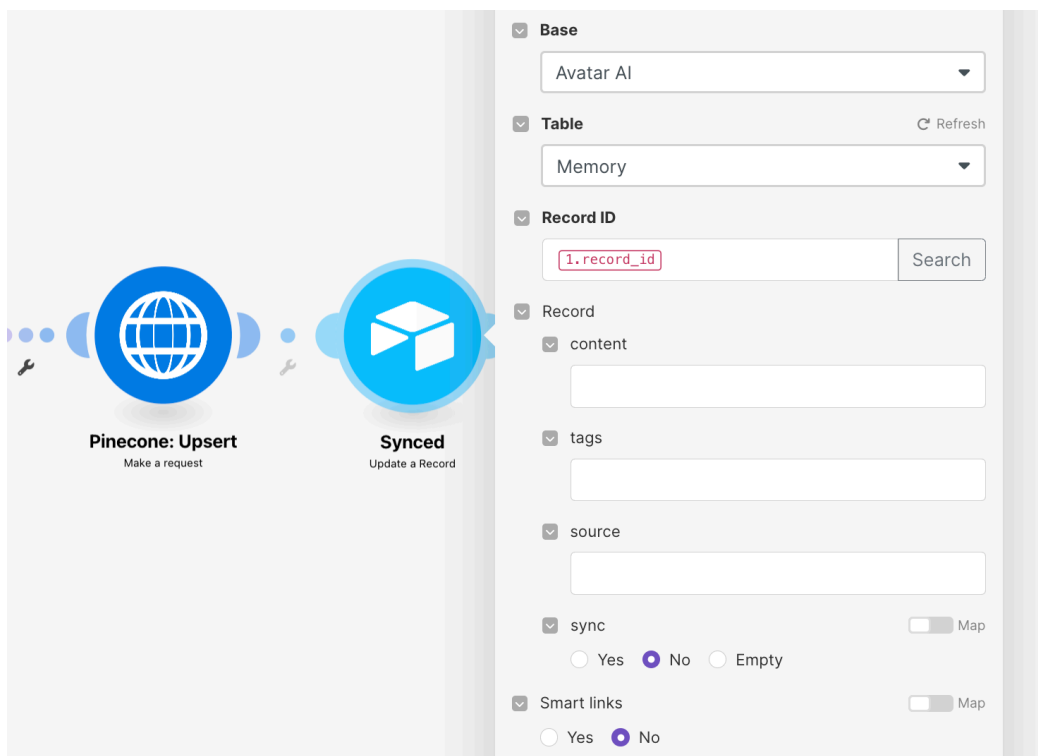
1. We need to send generated embeddings to the Pinecone, but first we have to create another JSON object. You can generate a data structure with this example: `{"vectors": [{"id": "", "values": [123]}]}`. And fill variables as shown on a screen below. Keep in mind that `id` in Pinecone is same as in Airtable.



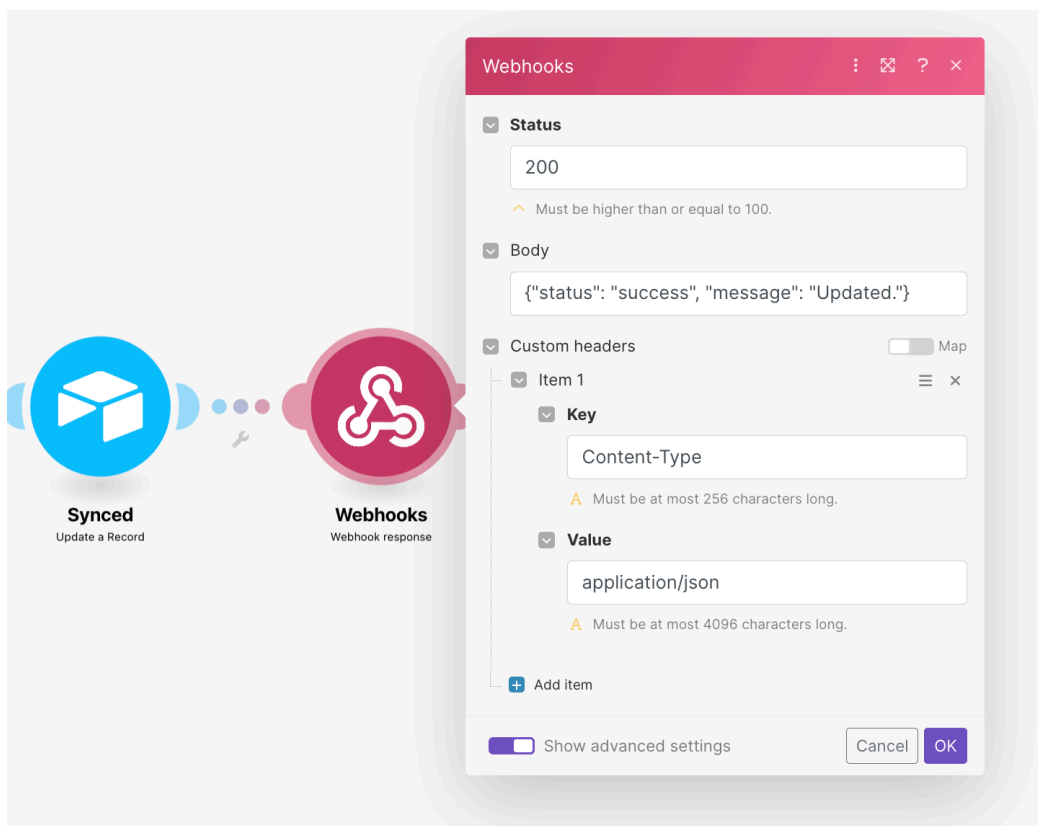
1. At this step you need to send information to the Pinecone with HTTP module. URL should be set to:
`https://YOUR_PINECONE_URL/vectors/upsert`. Also make sure to include headers (Api-Key and Accept) and JSON String as a body.



1. Everything is done, so we have to uncheck the checkbox in Airtable to make the row disappear from the "To Sync" table.



1. And the last step will be a Webhook response that look like this:

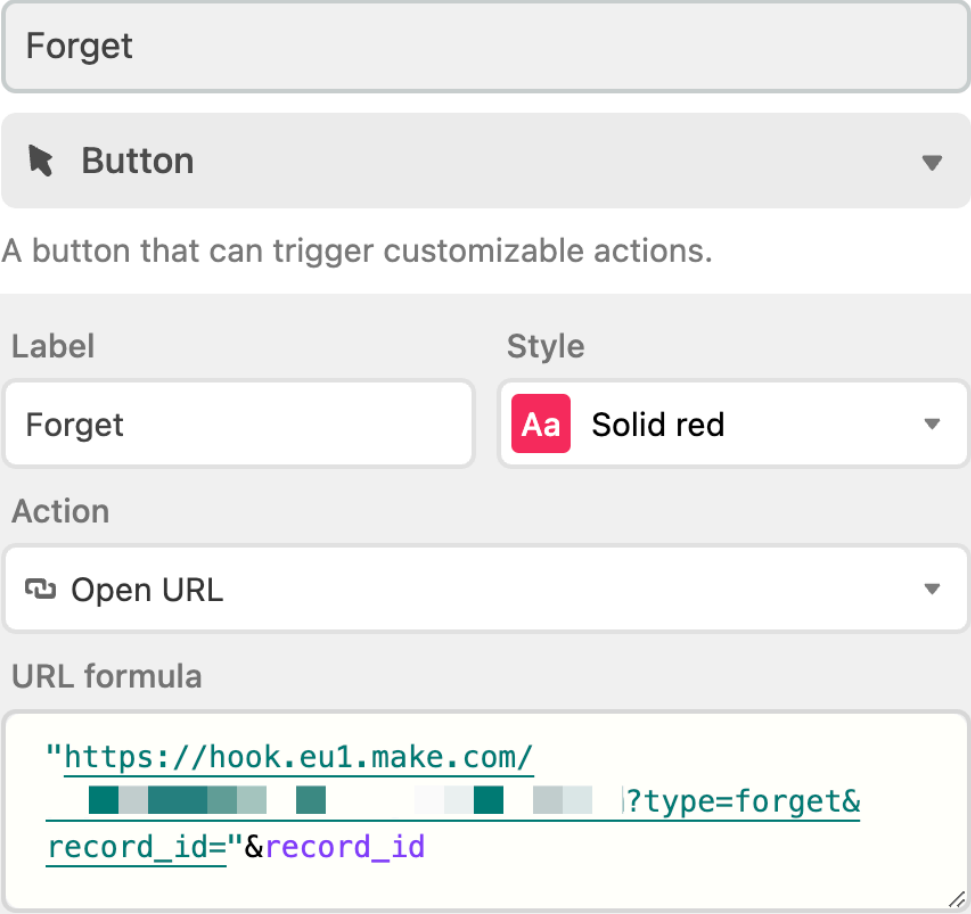


At this point, our Avatar can remember any text-information that appears in the "To Sync" table in Airtable.

I don't know if you can see the whole logic here, but what we do is store data both in Airtable (plain text) and Pinecone (embeddings).

Forgetting

Before we move on to the last step, we need to create a logic for forgetting information. It will be triggered manually with a button "Forget" in Airtable. Edit it's type like so:



The image shows the configuration interface for an Airtable button. It consists of several sections: a top bar with the label "Forget", a dropdown menu set to "Button", a description "A button that can trigger customizable actions.", and a configuration panel. The configuration panel has three main sections: "Label" with a text input containing "Forget", "Style" with a color picker set to "Solid red", and "Action" with a dropdown set to "Open URL". Below the action dropdown is a "URL formula" section with a text area containing the URL: `"https://hook.eu1.make.com/" + record_id + "?type=forget&record_id=" + record_id`. The text area has a color-coded syntax highlighter.

Forget

Button

A button that can trigger customizable actions.

Label

Forget

Style

Aa Solid red

Action

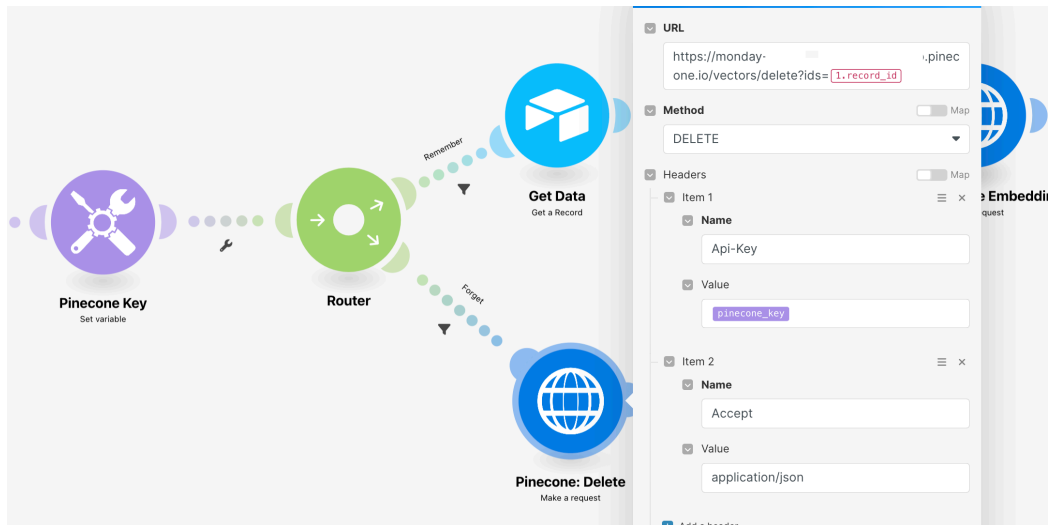
Open URL

URL formula

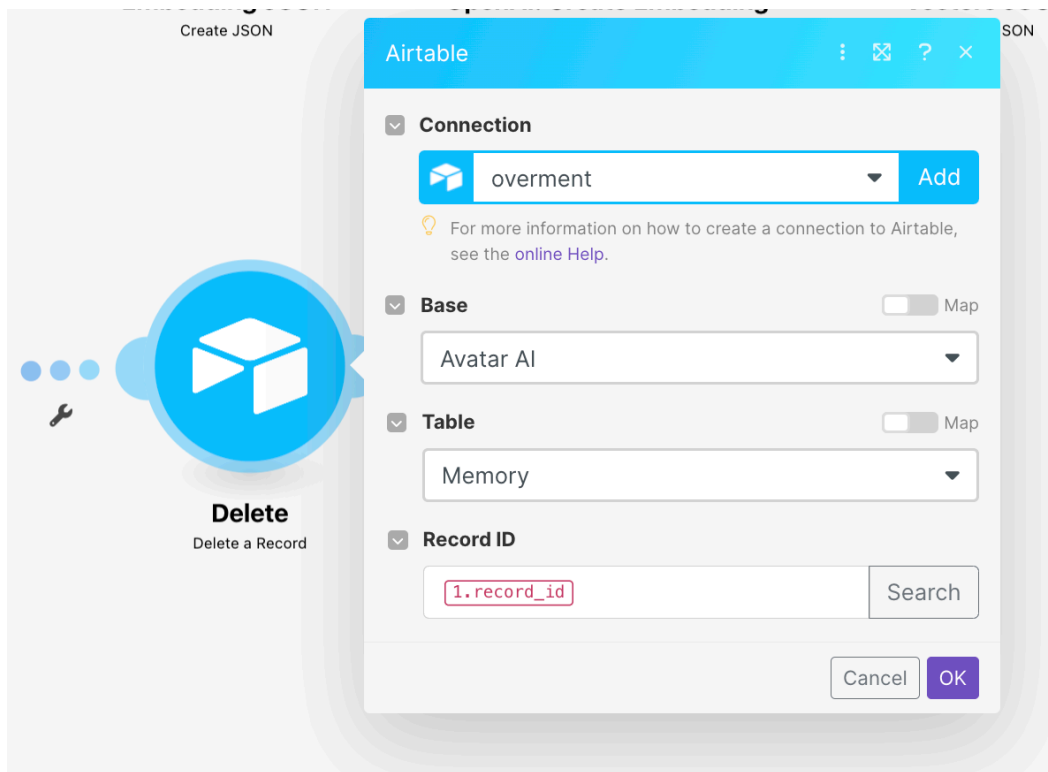
`"https://hook.eu1.make.com/" + record_id + "?type=forget&record_id=" + record_id`

Make sure to add your webhook address. After hitting this button, our scenario will be triggered with `type=forget` and `record_id`.

To our latest Router, add another path and make HTTP Request to the Pinecone like on image below. Make sure to include a filter to check if `type=forget`:



Right after this request, you can remove this record from Airtable:

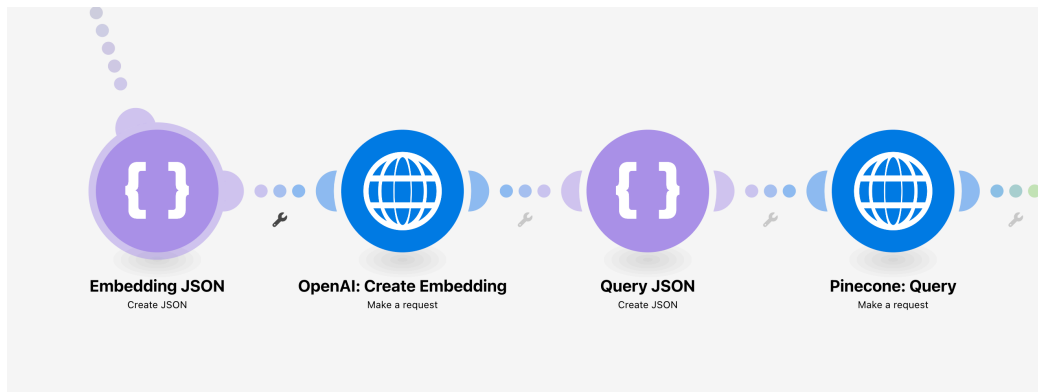


That's all. To delete and forget a given information, just hit the "Forget" button in Airtable, right next to it.

Context

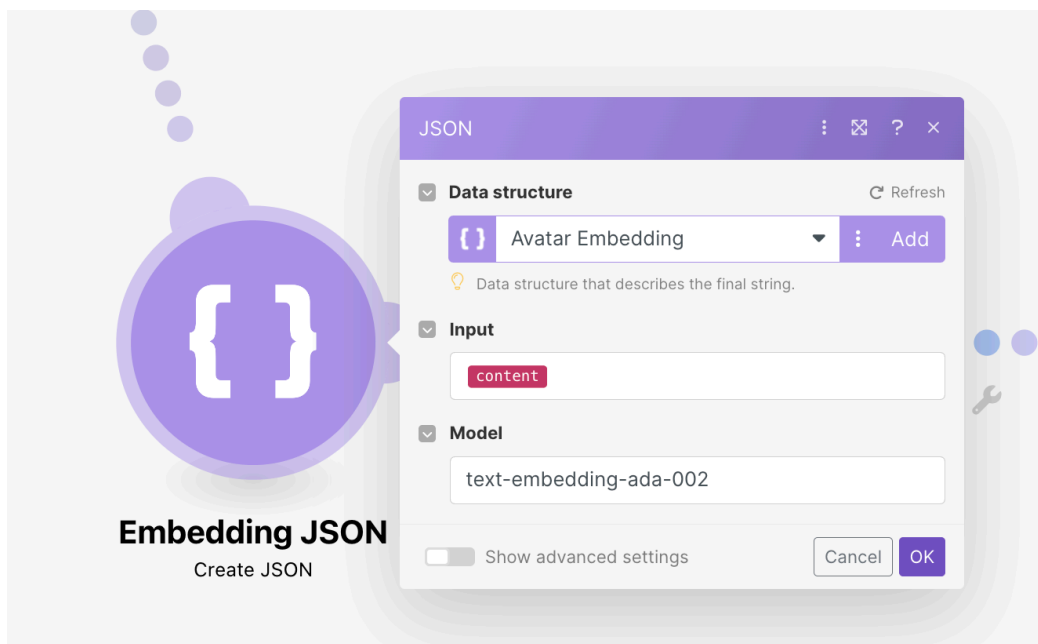
We've got everything we need to create last part of Avatar's logic. To write a response, our scenario has to:

- convert current message (`content`) to embeddings
- use those embeddings to query Pinecone database
- use IDs retrieved from Pinecone to query Airtable
- get current conversation
- generate a response
- send response
- save current conversation
- Converting the current message to embeddings and querying Pinecone will look the same as it did in a previous path.



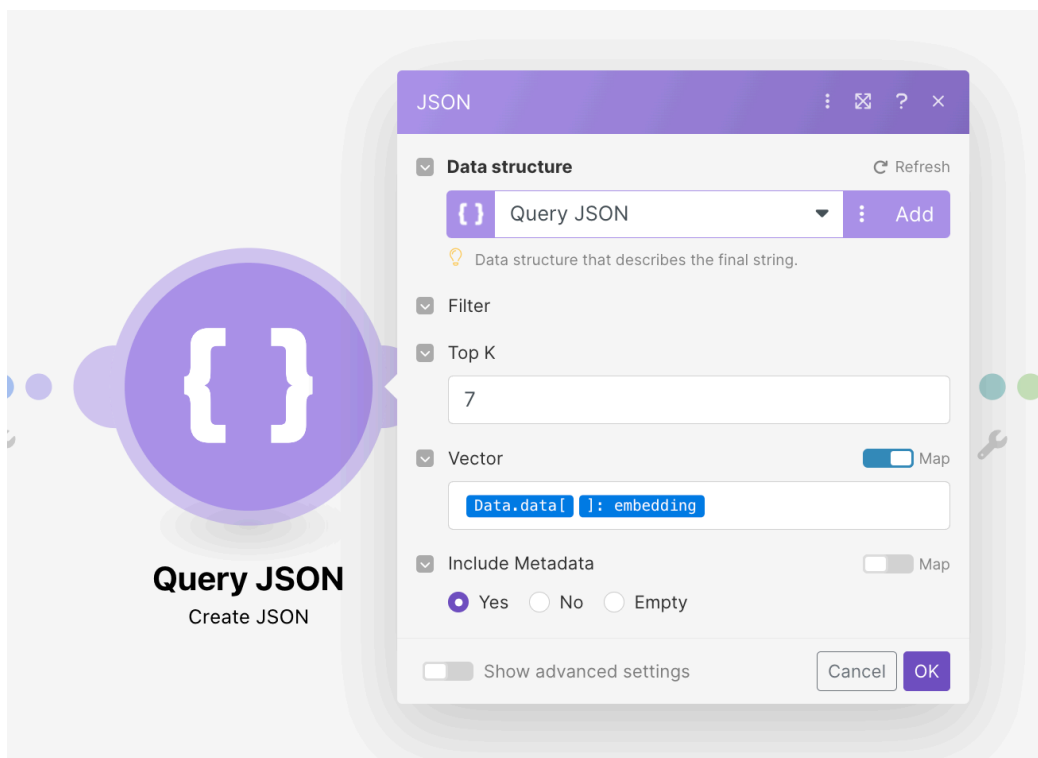
Embedding & Querying

To convert `content` to embeddings, pass it as input to the JSON and create embeddings, exactly as before.



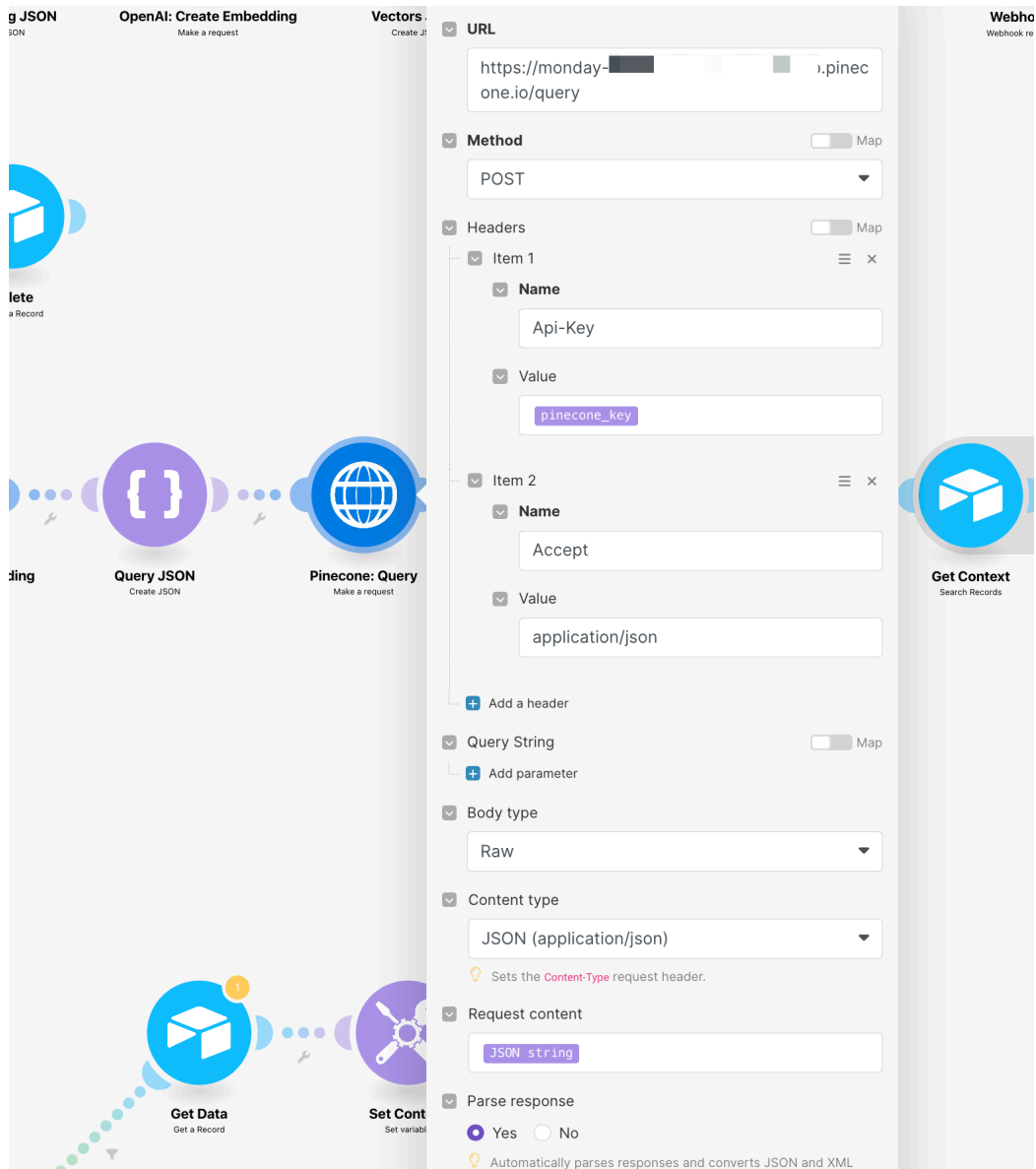
Embeddings

To query Pinecone we need a new JSON. You can create its structure like so based on this example: `{"topK":7,"filter":{},"vector": [123], "includeMetadata":true}`.

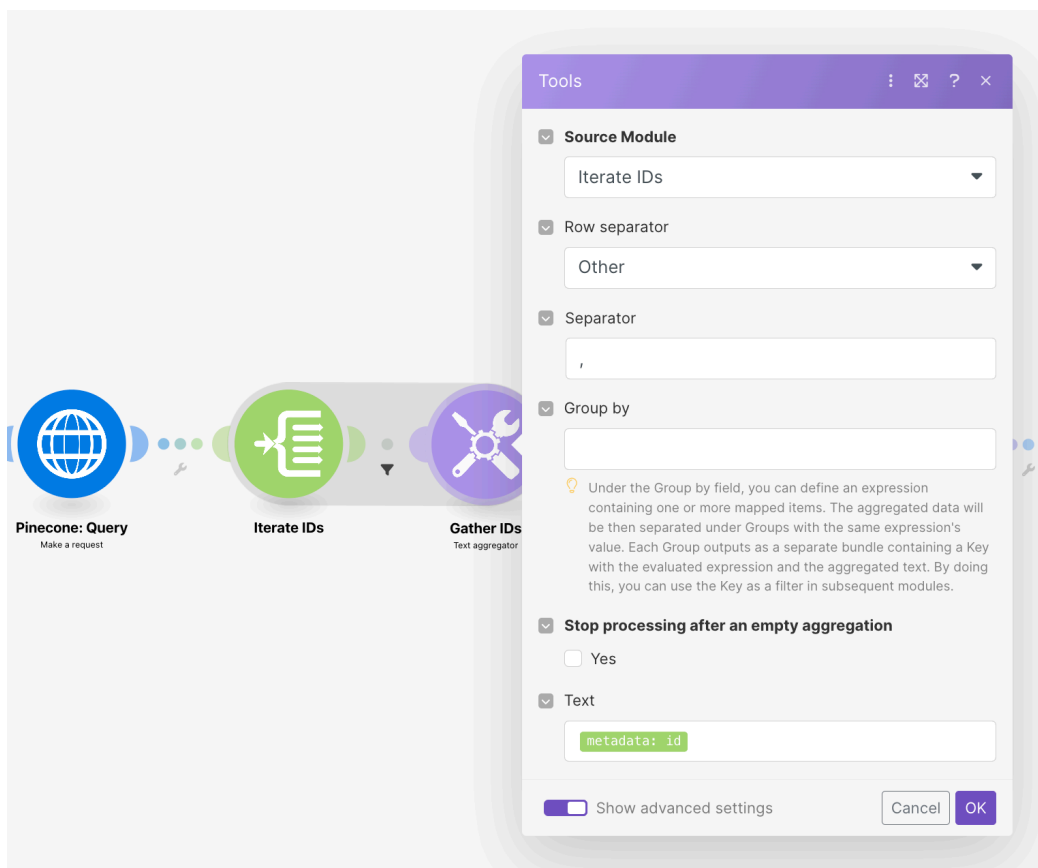


Querying

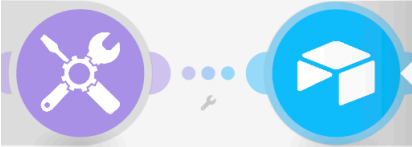
Request to the Pinecone needs to be sent to
`https://YOUR_PINECONE_URL/query` URL with settings like this:



1. Now we can use IDs returned from Pinecone to fetch Airtable records. Pinecone returned an array, so we have to use "Iterator" module to iterate through all of the elements and gather IDs.



1. With Airtable Formulas, we have to select only those records whose IDs are on a list from Pinecone.



Gather IDs
Text aggregator

Get Context
Search Records

Connection

overment Add

For more information on how to create a connection to Airtable, see the [online Help](#).

Base Map

Avatar AI

Table Map

Memory

Sort Map

+ Add item

View Map

Output Fields Map

- ☒ record_id
- ☒ content
- ☒ tags
- ☒ source
- ☐ sync

Limit fields in the output. Leave empty to retrieve all fields.

Formula

IF(SEARCH({record_id}, "text"), 1, 0)

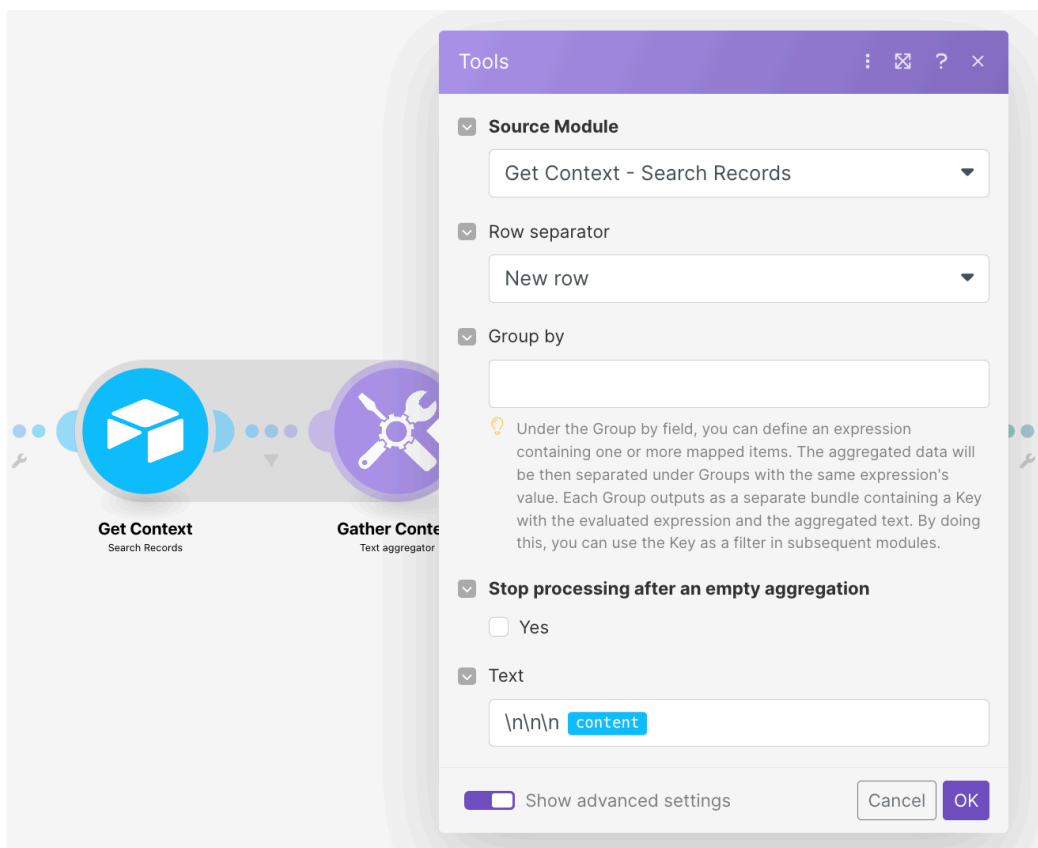
An Airtable formula used to filter records. The formula will be evaluated for each record, and if the result is not 0, false, "", NaN, [], or #Error! the record will be included in the response. You can find more information about the formula on [Airtable's website](#). For example, to only include records where Name isn't empty, pass in: NOT((Name) = "")

Limit

10

The maximum number of records to return.

1. All `content` fields have to be merged into a single string. We can do this by using the Text Aggregator again. Row separator needs to be set to "New row" but also, we need to add more detailed separator like "###" or "\n\n\n".



1. The last step of preparing our context will be fetching the last conversation. It will be empty at the start, but that's okay. Just make sure you limit the results to a single row; it's crucial, as it will make the rest of the scenario execute multiple times.

⚠ WARNING: Read a description above once again and really make sure that a limit is set to 1.


```

###
Your memory:
{{32.text}}
###
Conversation summary:{{if(41.summary; 41.summary; "[no history]")}}
###

- [Name]:{{1.content}}
- Answer:

```

Avatar behavior can be defined however you want, but first take a closer look at the template above. We have a very clear separation of "Description," "Memory" "Summary," and, lastly, the Question and Answer.

An example prompt may look like this: Keep in mind that the prompt needs to be as concise and detailed as possible; every word counts, because even at small scale it may generate noticeable costs.

```

As Avatar AI conversing directly with your friend, [name]. Respond to his latest message based on your memory, current conversation and knowledge about yourself, using complex, grammatically correct sentences, a natural tone, and if you cannot relate to the question or it is not included in your memory, be honest and write "I don't know", asking a probing question. Draw conclusions and summarize the conversation to find out and understand what you are talking about. Pay special attention to the differences between your memory and current conversation. Be specific, polite, clear, honest and interested.

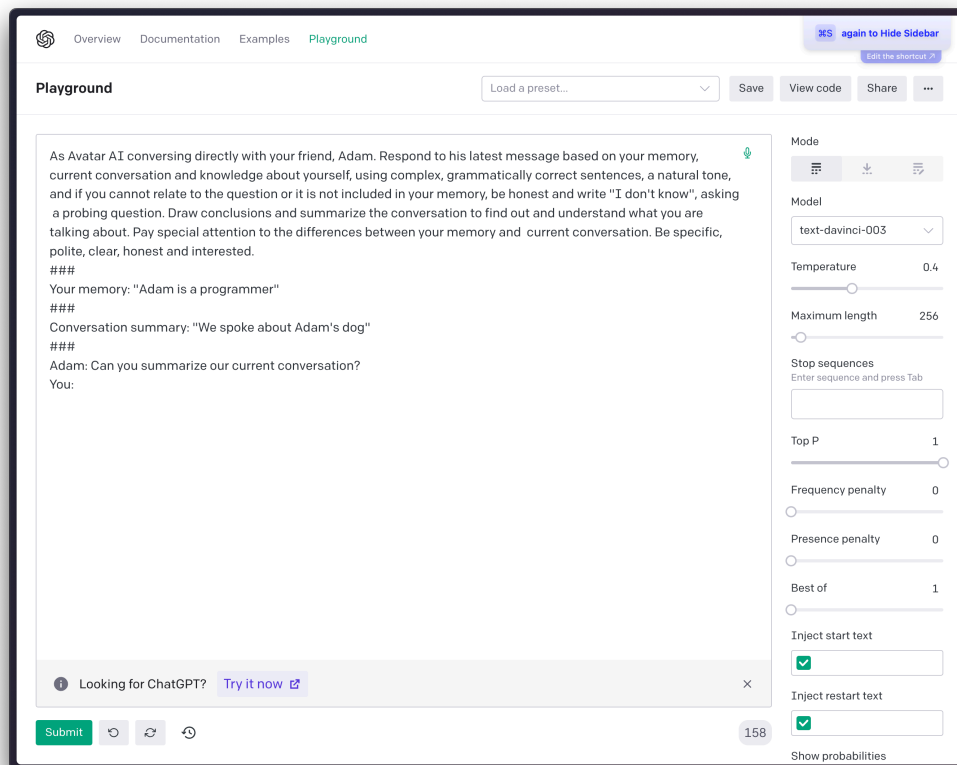
```

```

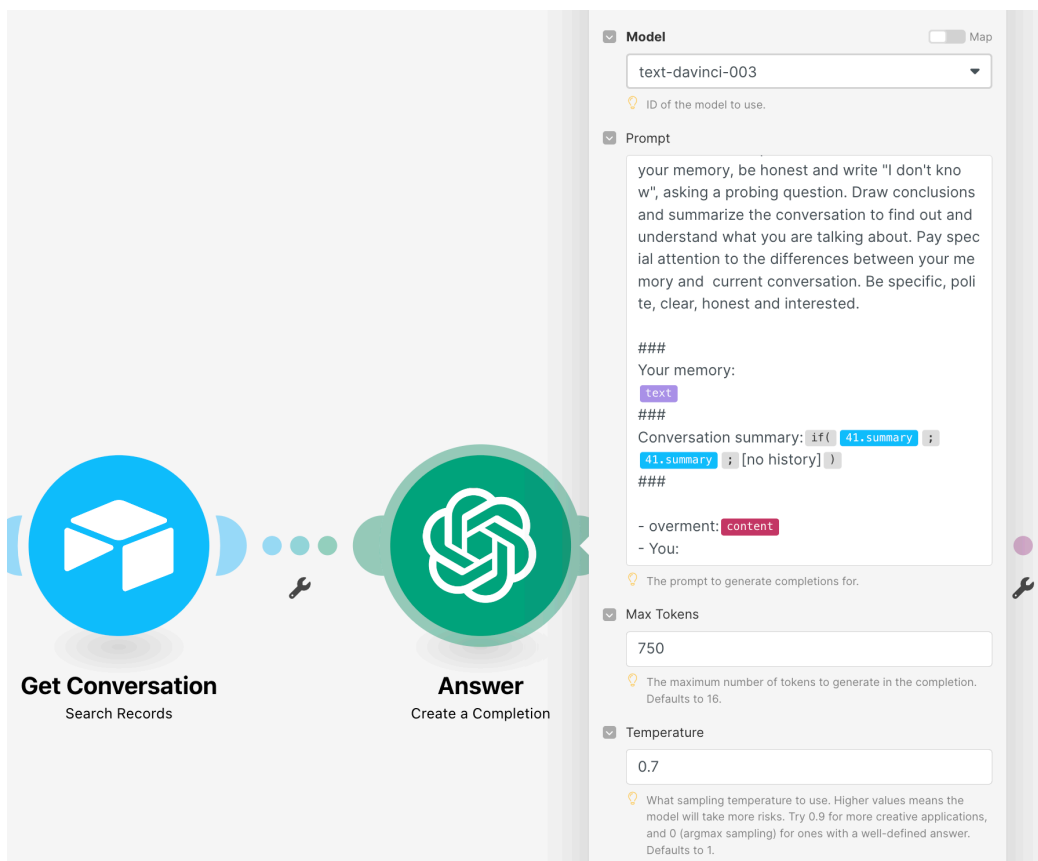
###
Your memory: "[name] is a programmer"
###
Conversation summary: "We spoke about [name]'s dog"
###
[name]: Can you summarize our current conversation?
You:

```

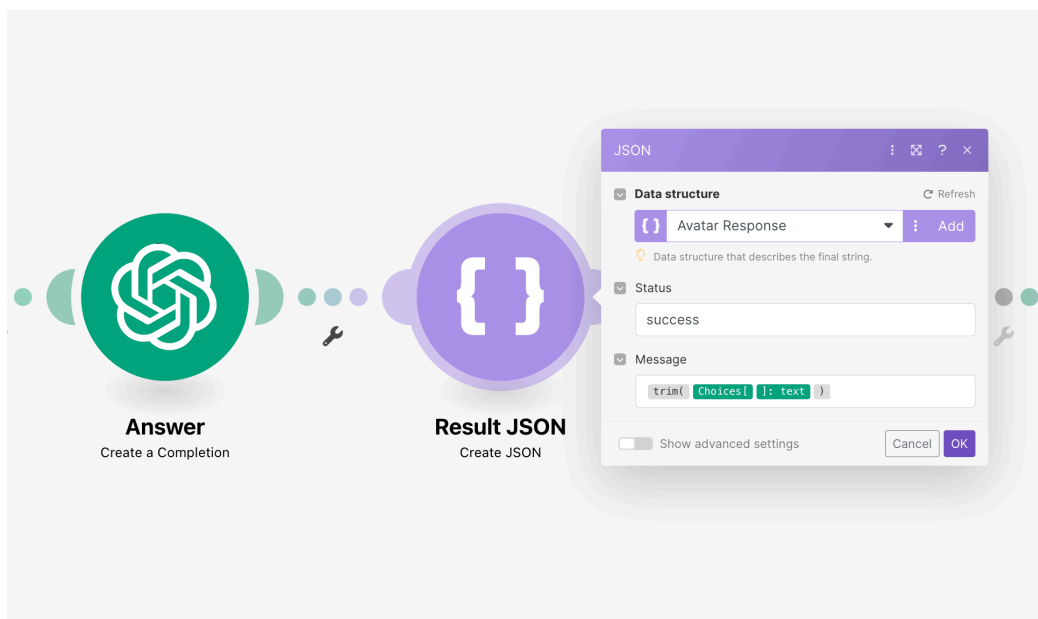
To test your prompt, you can use OpenAI Playground:



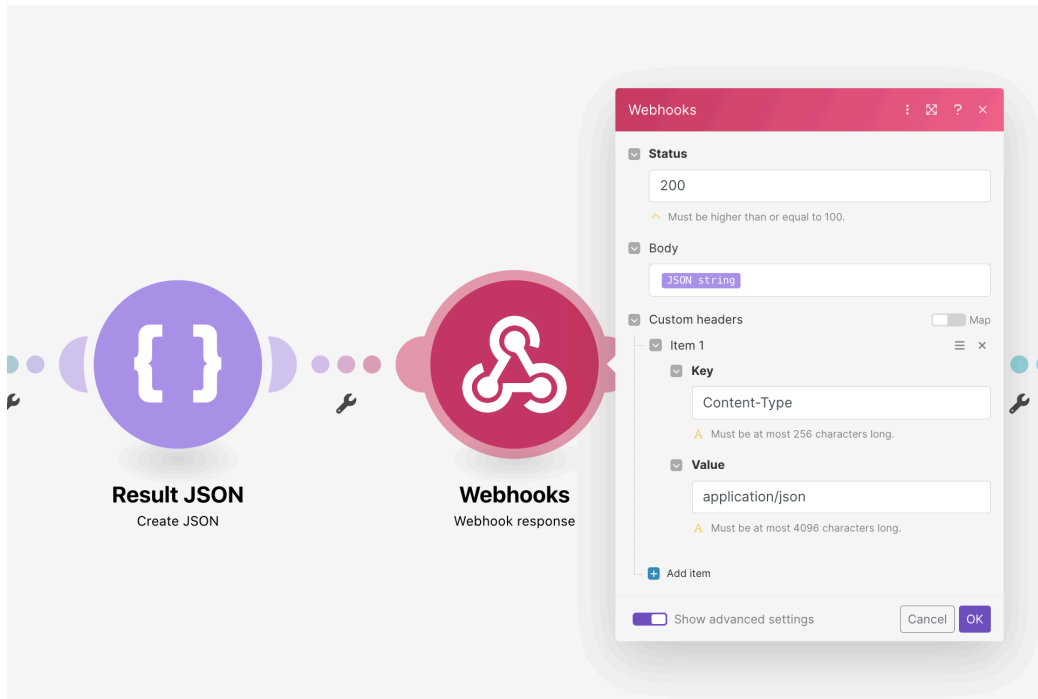
We can go back to our scenario. With native GPT-3 module, we can access OpenAI API and insert our prompt:



Based on a response from GPT-3 module, we need to create a JSON using this example to generate Data Structure: `{"status": "success", "message": "..."}`

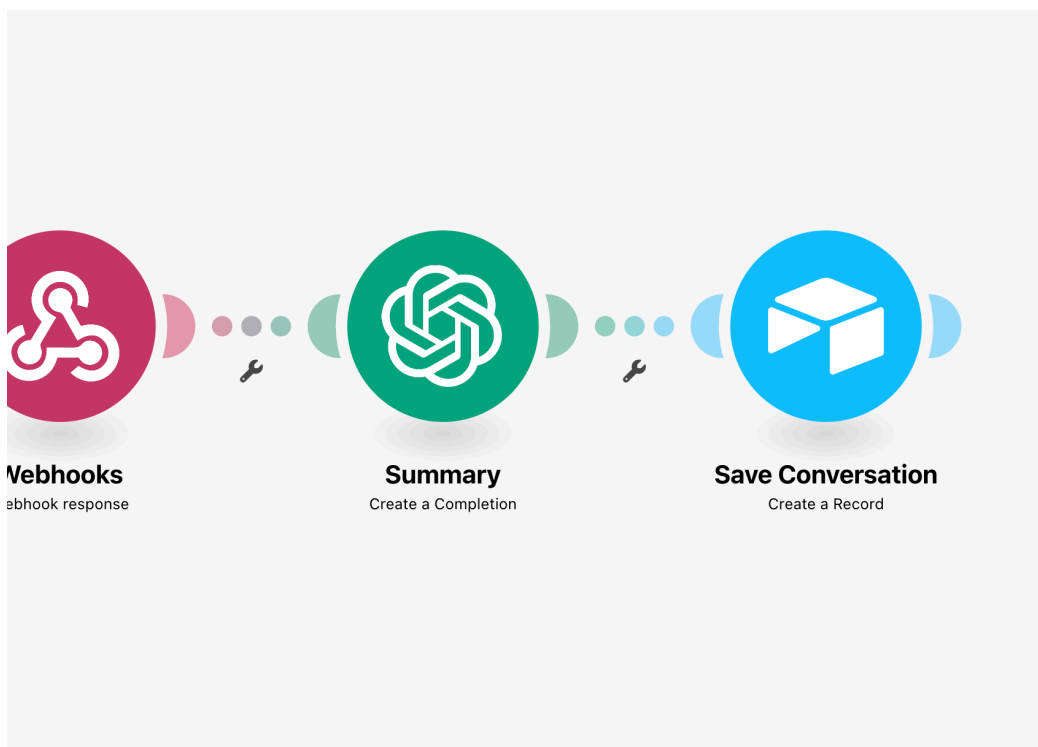


And lastly there is a Webhook Response with JSON body. Please make sure to use Advanced Settings and append "Content-Type" header set to "application/json".



Conversation Summary

Right now, we don't have any summaries of conversations. Therefore, after a response, we can use GPT-3 to generate a summary of the current dialogue. The result should be saved in Airtable.



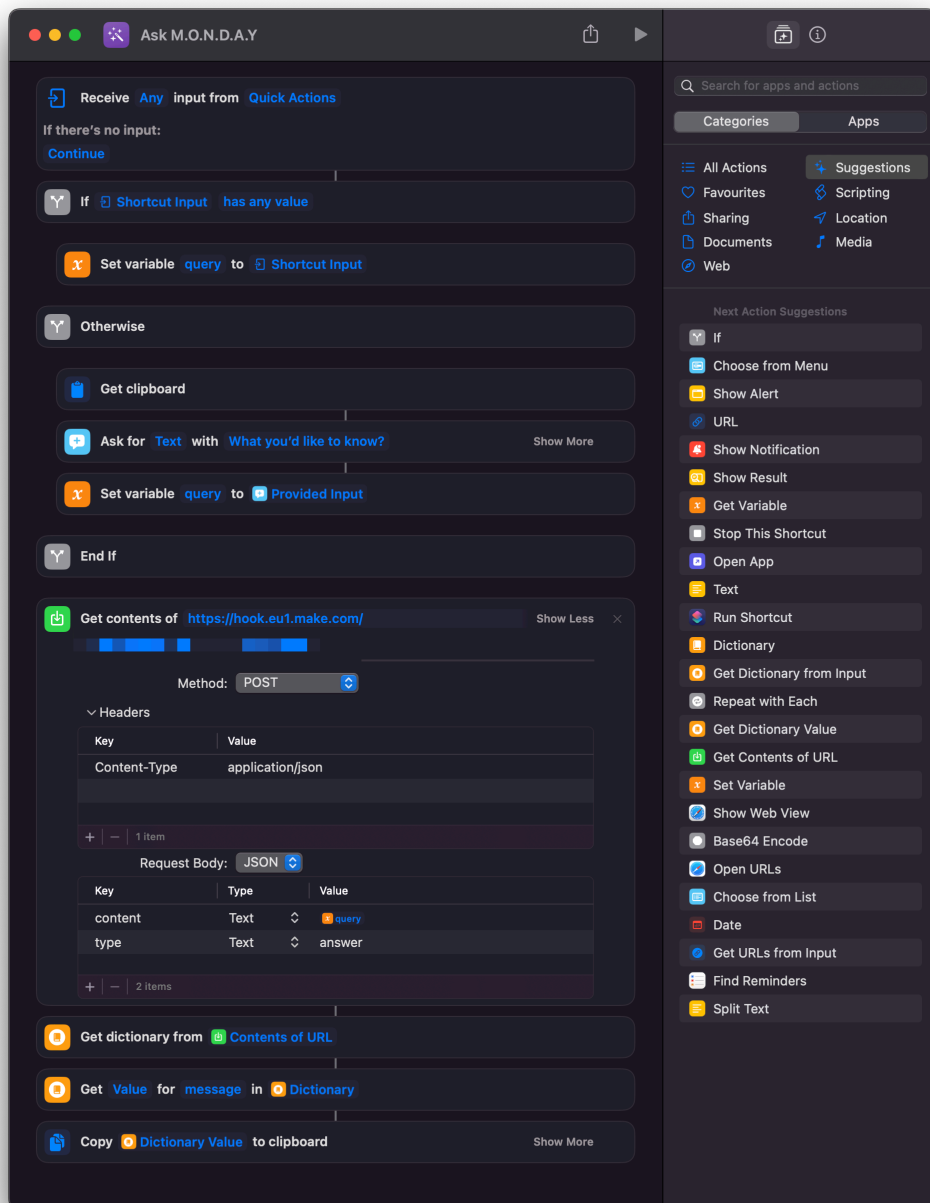
```
As AI Avatar I will summarize for myself, my conversation with
[name], very precisely marking which words he spoke and which I said.
{{if(41.summary; "I will combine the facts from the summary of the
previous conversation to create an updated version.
###
Previous summary:
" + 41.summary)}}
###
Latest Dialogue:
[name]: "{{1.content}}"
Avatar: "{{31.choices[].text}}"
###
Summary of Conversation:
```

At this point, we have everything. Our Avatar can both remember and forget; we can converse with her, and she will remember our conversation.

There is one more thing left to do: an interface. It may be anything what can send HTTP requests to our webhook, for example Shortcuts app.

Shortcuts & Siri

Do you know the Shortcuts app? You can create macros that can be triggered with a keystroke or Siri voice command. To interact with our Avatar, we can build macro like this:



As you see, first I get a value from Shortcut Input or from a Clipboard. Then I make HTTP POST request to my scenario in make. A response is being parsed and copied to the clipboard (and also returned from this Shortcut. It's

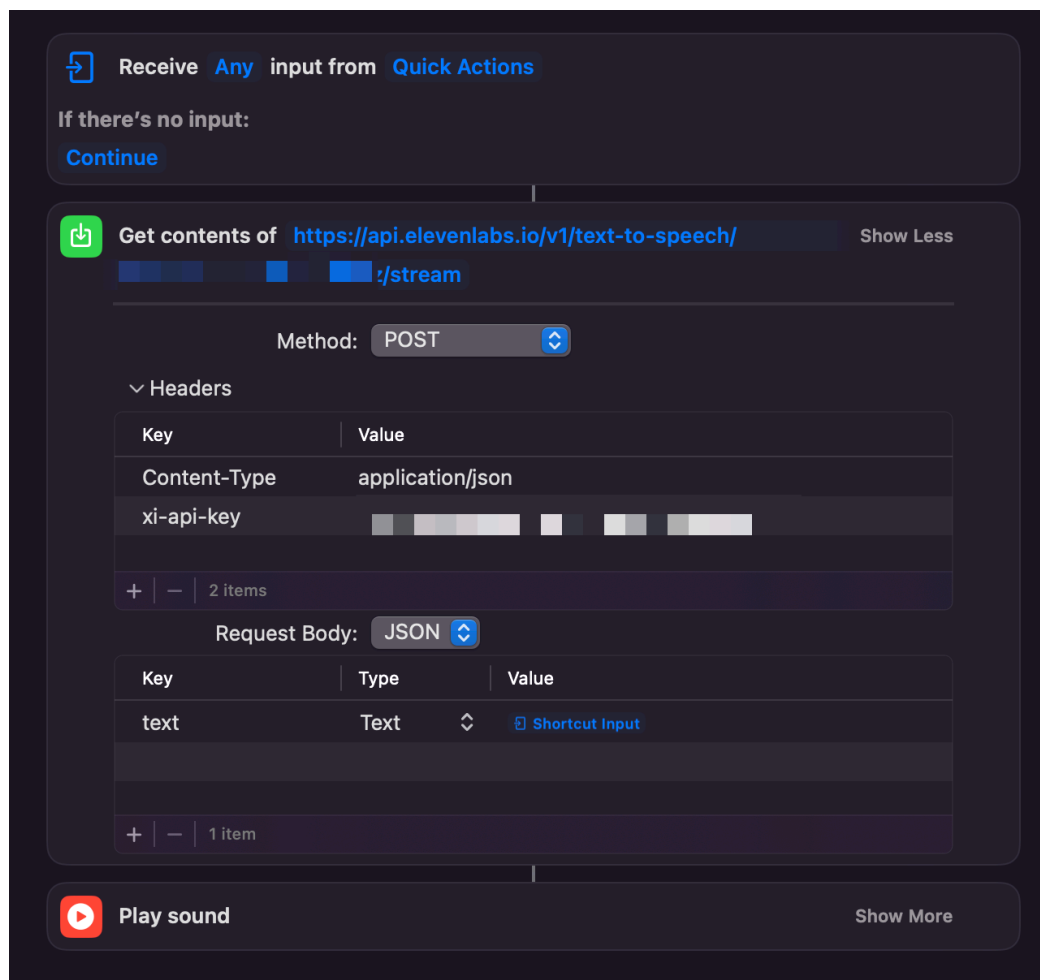
important and I'll show you. why).

Macros like this can be triggered with a keyboard shortcut on your Mac or with a gesture on your iPhone. You can configure it however you want, as you now have all the main concepts.

Voice Interaction

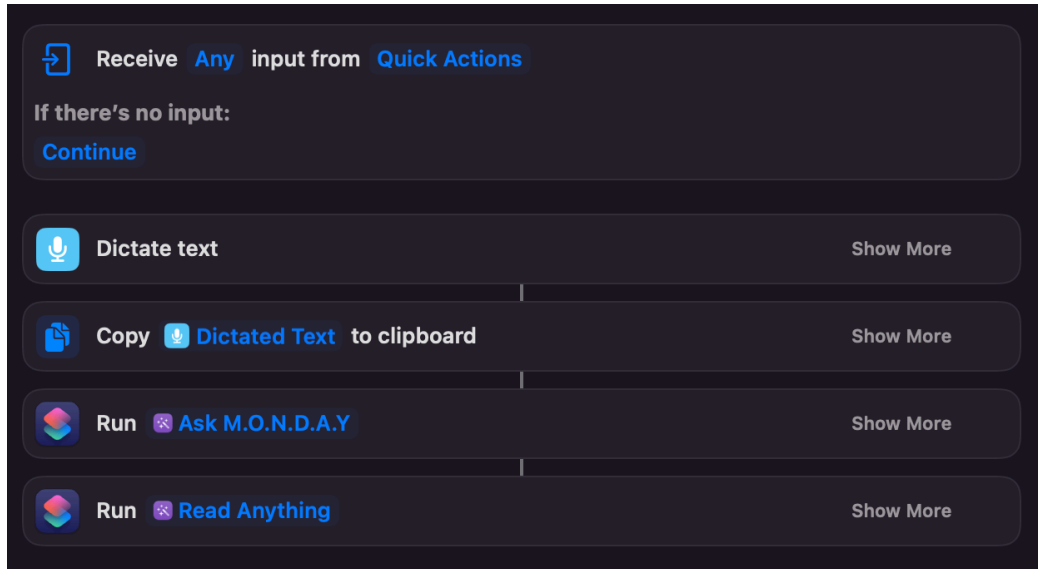
ElevenLabs is awesome! Based on a 30-60 second audio of your voice, it can read any text with it. But the best part, as always, is an API, which you can use in Shortcuts app.

With a very simple macro like this:



You can automatically convert text to audio. Of course, access to this API is paid, but it may be more than enough for some interactions.

Anyway. This macro above won't be triggered directly, but through another Shortcut. I have a macro like this:

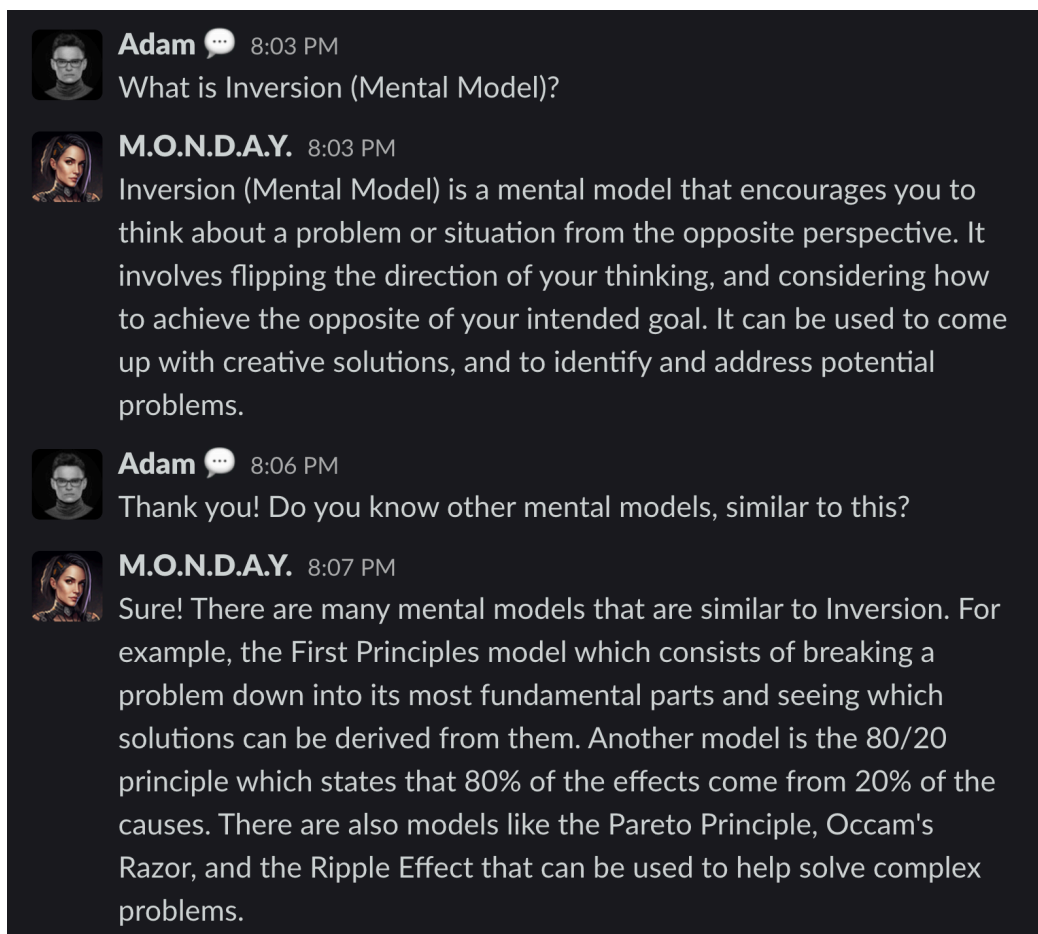


It's a very simple action that will allow me to dictate text and then trigger both previous macros to get a result and read it.

The fun part is: you can trigger Shortcuts with your voice, using Siri. But this time, you'll speak with your AI Avatar!

Result

You may be curious as to how this all works. Here is my conversation with M.O.N.D.A.Y.:



I can communicate with her using both Siri Shortcuts and Slack. As I said, you can use any interface you want; all you need to do is send a simple HTTP request.

On the screenshot above, you can clearly see that she can answer my questions and is aware of the context of our conversation. In the second message, I just said, "similar to this," and she was aware of what I was asking.

In fact, the possibilities are almost endless here. I'm pretty sure that you'll need to make some changes to the examples of prompts I gave, but still, you've got a core idea, so you can use it.

Summary

Creating an AI Avatar using no-code tools may seem complicated, but everything you see in this article was created by me in a couple of days. There's no doubt that this approach could be better. Right now we're using only "in-

prompt learning", which is slow and cost more. But also it's way more flexible than fine-tuning for example. Moreover, M.O.N.D.A.Y. is far more advanced than what you can see here. I will definitely share my future experiences, ideas, and techniques;

Just follow my profile.